Comparison Report

Code Improvement

Brief Introduction

Brief Intro to the updated code: After improving the code, I design five mode for the main function in the prediction system.

- Predict: Enter a Chinese name, and the model will predict the gender. You can choose between different models (Logistic Regression, Random Forest, LSTM) and also opt to view the prediction probabilities.
- Explore: This mode allows you to see the performance of different models on a split of the data, showing accuracy and a classification report for Logistic Regression, Random Forest, and LSTM models.
- Train: If you need to train the model with your dataset, this mode will initiate the training process and save the trained model for later use.
- Test: Here, you can test the performance of the already trained models on a separate test dataset and see the results in terms of accuracy and a detailed classification report.
- K-Fold Validation: Use k-fold cross-validation to assess the model's performance and save the validation reports for future reference.

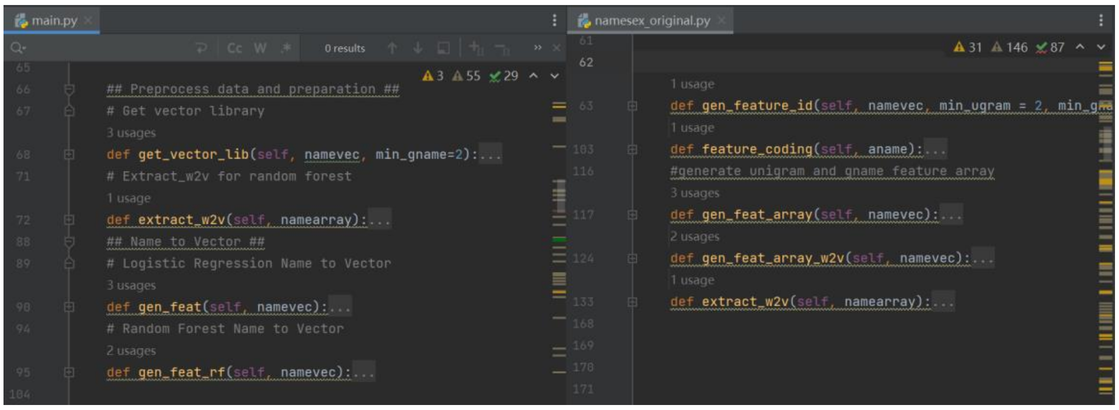
Beyond the model: While using various models to build the prediction model, I also found that there are some more neutral or specific meanings of Chinese names that are difficult to predict, such as '胜男' and '招弟', which represent female names that are easily predicted as male by the model. To solve this problem, I came up with two ways, one is to build a statistical library to analyze this kind of names individually, and the other is to use a large language model to process the names with other information. I currently did not explore this issue in depth due to time constraints.

Data preprocessing

a) **Major effort:** I built four new functions to replace the former data preprocessing part, which offers an enhanced approach to extracting word embeddings and generating features from the given names vector. By refining the structure and logic of the functions, the updated version provides a more streamlined and efficient solution.

b) **Advantages:** The main enhancements are improved readability, improved vector conversion efficiency, maintaining modularity and Reusability.

c) **Related Code:**



main.py

```
## Preprocess data and preparation ##
# Get vector library
3 usages
def get_vector_lib(self, namevec, min_gname=2):...
# Extract_w2v for random forest
1 usage
def extract_w2v(self, namearray):...
## Name to Vector ##
# Logistic Regression Name to Vector
3 usages
def gen_feat(self, namevec):...
# Random Forest Name to Vector
2 usages
def gen_feat_rf(self, namevec):...
```

namesex_original.py

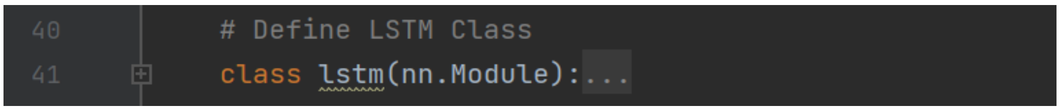
```
1 usage
def gen_feature_id(self, namevec, min_ugram = 2, min_g
1 usage
def feature_coding(self, aname):...
#generate unigram and gname feature array
3 usages
def gen_feat_array(self, namevec):...
2 usages
def gen_feat_array_w2v(self, namevec):...
1 usage
def extract_w2v(self, namearray):...
```

Model Improvement

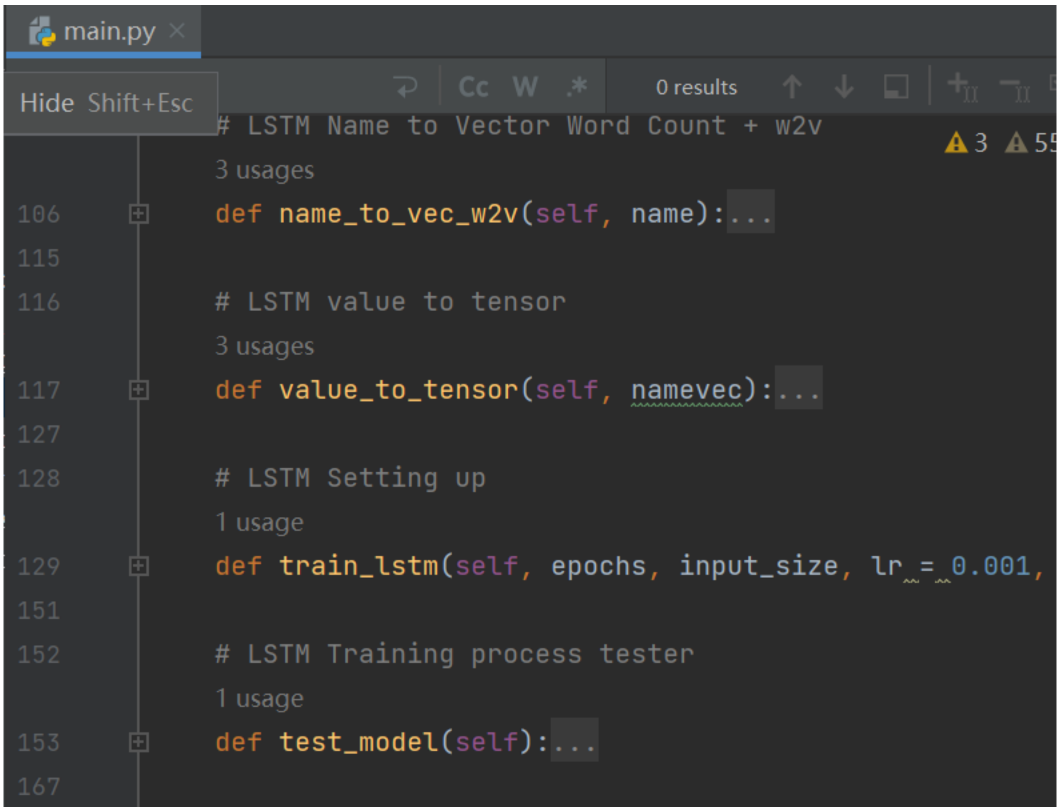
a) **Major effort:** I introduce LSTM as a new model. The whole process includes:

1. Name to Vector (Conversion Process): in the 'name_to_vec_w2v' function, I combine word count vectors and word2vec embeddings to form a comprehensive representation for the name.
2. Vector to Tensor: In the 'value_to_tensor' function, I have ensured that names are properly transformed into tensors suitable for feeding into an LSTM network. This involves padding sequences to match the length of the longest name and stacking them appropriately.
3. Model Establishment: LSTM architecture is designed in the 'lstm class' and setting up in the 'train_lstm'. It starts with dropout regularization, followed by an LSTM layer. The output from the LSTM is then processed through two linear layers with a ReLU activation function in between. The final output goes through a LogSoftmax function, making it apt for classification tasks. For each epoch, the model is trained on batches of name tensors and then validated using the test_model function to check accuracy.
4. Fine-tuning the Model: I also fine-tuned the model to determine the best combination of batch size, learning rate, and epochs to achieve optimal performance.

b) **Related Code:**



```
# Define LSTM Class
class lstm(nn.Module):...
```



main.py

Hide Shift+Esc

0 results

```
    # LSTM Name to Vector Word Count + w2v
    3 usages
    def name_to_vec_w2v(self, name):...

    # LSTM value to tensor
    3 usages
    def value_to_tensor(self, namevec):...

    # LSTM Setting up
    1 usage
    def train_lstm(self, epochs, input_size, lr = 0.001,

    # LSTM Training process tester
    1 usage
    def test_model(self):...
```

Other Enhancement & Main Function

a) **Major effort:**

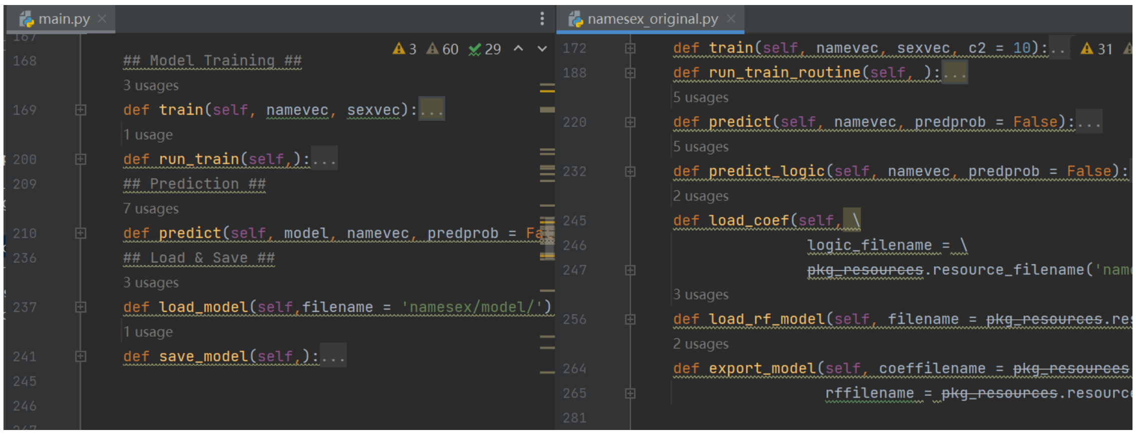
1) Consolidation of Predict Functions:
   - Before: Separate prediction functions were used for different models (predict for Random Forest and predict_logic for Logistic Regression).
   - After: A unified predict function was introduced where the model type (random forest, logistic regression, or LSTM) is provided as an argument. This reduces redundancy and enhances clarity.
2) Refactoring Model Loading and Saving:
   - Before: Separate loading functions (load_rf_model, load_coef) were used for different models, and there was commented-out code which cluttered the overall structure.
   - After: Consolidated the model loading into a single function, load_model, and provided a centralized way to save all models with save_model.
3) Export Function:
   - Before: The export_model function was used to export coefficients and the Random Forest model separately.
   - After: Replaced by the more generic save_model function.
4) Refinement in File Handling:
   - Before: Loading and saving relied on pickle and were manually handling file types (like .pickle, .bin).
   - After: Transitioned to more specialized joblib's load and dump for Random

Forest and Logistic Regression while utilizing PyTorch's native saving for the LSTM model.

b) **Advantages:**

1. Enhanced clarity and simplicity.
2. Better model management.
3. Integration with advanced deep learning techniques.
4. Efficient and safer file handling methods.

c) **Related Codes:**



main.py

```
    ## Model Training ##
    3 usages
    def train(self, namevec, sexvec):...
    1 usage
    def run_train(self,):...
    ## Prediction ##
    7 usages
    def predict(self, model, namevec, predprob = Fa
    ## Load & Save ##
    3 usages
    def load_model(self,filename = 'namesex/model/')
    1 usage
    def save_model(self,):...
```

namesex_original.py

```
    def train(self, namevec, sexvec, c2 = 10):..
    def run_train_routine(self, ):...
    5 usages
    def predict(self, namevec, predprob = False):...
    5 usages
    def predict_logic(self, namevec, predprob = False):
    2 usages
    def load_coef(self, \
                  logic_filename = \
                  pkg_resources.resource_filename('nam
    3 usages
    def load_rf_model(self, filename = pkg_resources.re
    2 usages
    def export_model(self, coeffilename = pkg_resources
                     rffilename = pkg_resources.resourc
```

Performance Comparison

Note that even though the original document gives the ten-fold validation obtained by the original author's training, I still ran it locally with the code provided and obtained the following result:

Ten-fold Cross Validation

Logistic Regression - Original			**Logistic Regression - Aft**		
Metric	**Performance**	**Performance Std. Dev.**	**Metric**	**Performance**	**Performance Std. Dev.**
Accuracy	0.9044	0.0083	Accuracy	0.9011	0.004029
F1	0.9007	0.0086	F1	0.9009	0.004025
Precision	0.8962	0.0143	Precision	0.9007	0.004093
Recall	0.9057	0.0146	Recall	0.9014	0.003922
Random Forest - Original			**Random Forest - Aft**		
Metric	**Performance**	**Performance Std. Dev.**	**Metric**	**Performance**	**Performance Std. Dev.**
Accuracy	0.8872	0.0085	Accuracy	0.882	0.008574
F1	0.8832	0.0094	F1	0.8818	0.008676
Precision	0.8765	0.0174	Precision	0.8818	0.008746
Recall	0.8904	0.0129	Recall	0.8825	0.008213
LSTM					
Metric		**Performance**		**Performance Std. Dev.**	
Accuracy		0.8995		0.006304	

F1	0.8993	0.006423
Precision	0.8994	0.006191
Recall	0.8998	0.006799

- **Logistic Regression:** The updated Logistic Regression model shows a slight decrease in accuracy from 0.9044 to 0.9011 but comes with a decrease in the standard deviation, indicating more consistent performance across different folds. The F1 score saw a minor increase, precision improved notably, and recall slightly decreased, all with reduced variability, suggesting the model's robustness has improved.
- **Random Forest:** For the Random Forest model, there's a slight decline in all performance metrics (accuracy, F1, precision, and recall) after the update. However, the standard deviations remain roughly consistent, indicating that the model's performance consistency has not significantly changed post-update.
- **LSTM:** The introduction of an LSTM model to the mix yields competitive results. It achieves an accuracy and F1 score comparable to the updated Logistic Regression model but slightly lower than the original. However, it shows less variability than both the original models, highlighting a more stable performance across the cross-validation folds.
- **Summary:** The updated models demonstrate improved stability in performance, as indicated by the lower standard deviations across metrics. Although there's a marginal decline in accuracy for the Random Forest model, the increased precision in the Logistic Regression model is noteworthy. The LSTM model introduces a reliable balance between performance and consistency.

Test Result

In addition, I tested the model with the test dataset provided by the original authors with the following results:

Model	**Logistic Regression - Ori**				**Logistic Regression - Aft**			
Metric	**precision**	**recall**	**f1-score**	**support**	**precision**	**recall**	**f1-score**	**support**
Female	0.76	0.79	0.77	28	0.74	0.82	0.78	28
Male	0.92	0.91	0.91	74	0.93	0.89	0.91	74
accuracy	0.8724			102	0.8725			102
macro avg	0.84	0.85	0.84	102	0.84	0.86	0.85	102
weighted avg	0.87	0.87	0.87	102	0.88	0.87	0.87	102
Model	**Random Forest - Ori**				**Random Forest - Aft**			
Metric	**precision**	**recall**	**f1-score**	**support**	**precision**	**recall**	**f1-score**	**support**
Female	0.82	0.82	0.82	28	0.79	0.79	0.79	28
Male	0.93	0.93	0.93	74	0.92	0.92	0.92	74
accuracy	0.9019			102	0.8823			102
macro avg	0.88	0.88	0.88	102	0.85	0.85	0.85	102
weighted avg	0.9	0.9	0.9	102	0.88	0.88	0.88	102
Model	**LSTM**							
Metric	**precision**		**recall**		**f1-score**		**support**	

Female	0.81	0.79	0.8	28
Male	0.92	0.93	0.93	74
accuracy	0.8922			102
macro avg	0.87	0.86	0.86	102
weighted avg	0.89	0.89	0.89	102

The Logistic Regression model remains relatively stable with slight improvements in gender classification after the update. The Random Forest model's performance slightly declines with the update. The LSTM model presents a good balance of precision and recall, making it a competitive alternative to traditional models. For the accuracy in test set has decreased, it may be due to the following reasons: 1. This training only uses the name to predict the gender, if the embedding dimension is elevated it will result in excessive segmentation of the name and the model may be unable to capture the features, which will reduce the prediction accuracy 2. The test set data is too small, especially there are only 28 women's names, which can't reflect the model's accuracy.